

Collapse Safe and Scalable Acyclic Graph Generation using Scale Free Distributions

Sandeep Gupta[1]

Abstract—Most of the generators currently used to build acyclic graphs do so via random selection of edges. Unfortunately, the graphs thus generated inherently have a tendency to collapse, i.e., the resulting graphs will always have almost the same and predictable structure. Specifically, we show that if $O(n \log n \log n)$ edges are added to a binary tree of n nodes then with probability more than $O(1/(\log n)^{1/n})$ the depth of all but $O(\log \log n^{\log \log n})$ collapses to 1. Experiments show that the maximum depth is very small (less than 3) thereby reducing the DAG to almost a bipartite graph. Their irregularity, as measured by distribution of length of random walks from root to leaves, is also predictable and small.

In this work, we develop an approach for generating directed acyclic graphs (DAGs) that have “truly irregular” structure, non-trivial depth distribution, and scale to multi million nodes. The generator provides the user knobs to control the structure and shape of the resulting DAG. Experiments are presented for graphs of size 2^{23} and varying edge densities, and in and out degree distributions.

I. INTRODUCTION

Data generators play an important role in algorithm design and optimization engineering. They are an important tool for modeling, benchmarking, scalability analysis, and cost estimation. In the last decade with the unprecedented growth in Internet, WWW, and, social networks the need for generators that produce graphs reflecting structures of these domains became prominent and has been an active area of research.

It was discovered in [11], [15] that such networks are fractal in nature (scale free) and can be described via physical phenomena of “rich gets richer” or “preferential attachment” [1]. The mathematical model that produces such fractal graphs is based on R-MAT [3] or Kronecker Product [16] and is the central theory underlying the scale free generators. A large body of work exists that utilizes either of the mathematical models to develop scalable scale free graphs [3], [19]. These works have contributed significantly towards the development of network protocols, algorithms, and, architecture design.

The community has paid little attention to generation of acyclic graphs. Acyclic graphs, much like scale free graphs, appear in many areas of computation and engineering. Knowledge representation, binary decision diagram, dependency graphs, semantic web, and, binaries of computer programs are a few examples. In the field of life-sciences and bio-informatics, such structures are used to create ontologies that represent the compendium of factual information. Advancement in bio-informatics and life-sciences research lead to an

almost rapid increase in the number of ontologies. Unlike social networks and WWW, the workloads in life-sciences and knowledge engineering disciplines are much more complex and include reachability and pattern queries, and, lowest common ancestors [8]. It is important for the database and the computing world to be able to develop algorithms over such workloads to better address the needs of the domain science. Graphs generators that produce realistic data sets would be indispensable for this exercise.

The size of acyclic graphs used in real world is not yet as large as the social-network graphs (upwards of multi billion edges) but they can potentially be many orders of magnitude larger than those present currently. This is particularly true in life-sciences, bio-informatics and knowledge representation domain. This is because disparate ontologies can reference entities across each other. For example, the Gene Ontology, has been constructed by combining ontologies of sub-species. Another ontology, Unified Medical Language System (UMLS) which maps the terminology of 60 different biomedical source vocabularies currently consists of one million biomedical concepts and five million concept name [14]. Hence, in order to drive and evaluate the next generation of information engines and tools, a scalable graph generator that can build acyclic graphs of varying characteristics is of immense importance.

Most of the generators currently used to build acyclic graphs do so via random selection of edges. Unfortunately, the graphs thus generated inherently have a tendency to collapse, i.e., as we scale the number of nodes and edges, the depth of the DAG exponentially decreases. Furthermore, the resulting DAG has the same predictable structure. We measure the property of “structural richness” by observing the distribution of random walks from source to leaves. A DAG with a rich structure would have a varied set of random path lengths. Figure 1 show this distribution for a randomly generated DAG. As we can see that the path length varies in a very narrow range (0–30) and that there is high concentrated of values in a even narrower range (10–15). The implication is that almost any path in a randomly generated dag, irrespective of the choice of random generators, will have path length of approximately 10 – 15. The depth distribution and degree distribution show similar concentration of values (see section IV-A).

Hence, such schemes can only generate graphs that are meaningful at small scales or have average degree of less than 5 ([4], [12], [21], [2]). Moreover, these schemes provide no control over the structure, such as fan in and fan out degree distributions, of the resulting acyclic graph. An approach with more control on the output graph was proposed in [20] but the approach does not scale due to the extensively large size

* San Diego Super Computing Center (SDSC), University of California, San Diego, California 92122 USA. Email:sandeep@sdsc.ucsd.edu

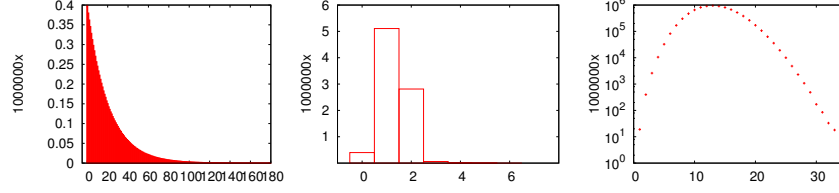


Fig. 1: Distribution of various properties in a randomly generated graph. Although a random (uniform) number generator is used to build the graph, all its properties are highly skewed and concentrated within a narrow range. Most importantly the depth distribution collapses to a few small values.



Fig. 2: Drawing of 128 node and 611 edges graph generated by the random generator using the Graphviz graph layout toolkit [9]. The aspect ratio of the figure is determined by graphviz based up on the structure of the graph. The small ratio of the figure clearly indicates that the graph has collapsed more so when compared with the non-degenerate (and aesthetically sound) height to width ratio of the drawing of the graph generated using our method (see figure 3).

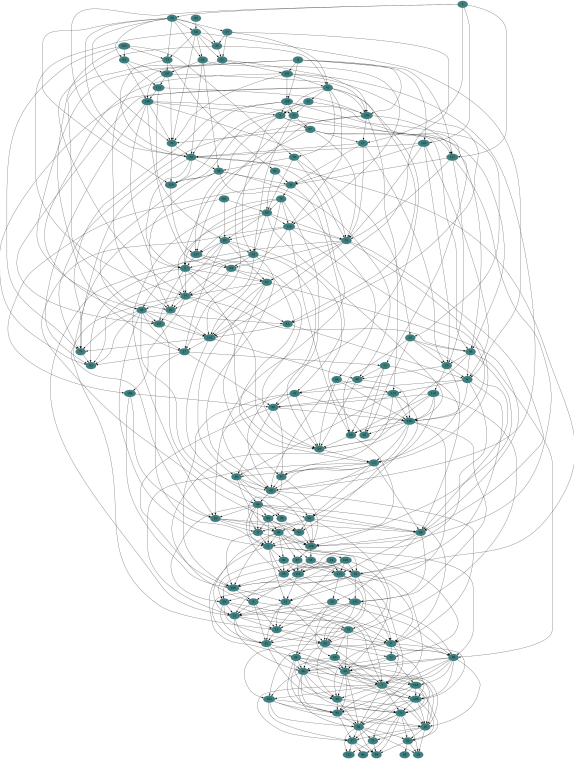


Fig. 3: Graphviz drawing of 128 node and 556 edges graph generated by our generator. The height to width ratio of the figure is almost one which is a further evidence that the graph has non-degenerate structure.

of the linear systems involved.

The purpose of this paper is to propose a generator for acyclic graphs. It addresses three challenges central to acyclic graph generation:

- Generate graphs that have “rich structure” and non-trivial

depth distribution¹.

- Scale generation of graphs to multi million nodes so that they represent real world graphs to be dealt with in near future
- Provide knobs to control the structure and shape of the resulting DAG.

We analytically and experimentally show that using random generation produces degenerate graphs, i.e., the resulting graphs always have almost the same and predictable structure. The maximum depth is very small (less than 3) thereby reducing the DAG to a bipartite graph. The irregularity, as measured by distribution of length of random walks from root to leaves, is very small. We demonstrate that just as scale free distribution was essential for structure of interaction graphs, this distribution is also crucial for generating acyclic graphs having rich structure.

The edge generation algorithm in our work is local and greedy. It performs ‘best effort’ to find the most suitable graph, given the input parameters. This allows the generator to produce very large graphs, up to 2^{27} nodes (experiments in this paper are discussed over graphs with 2^{23} nodes, larger graphs can also be generated). Unlike previous schemes, it provides more control to describe the structure of the resulting DAG. It takes as input a set of distribution functions that describe various DAG characteristics such fan-in, fan-out, and depth distributions. It then generates a stochastic DAG whose characteristics match with the input distributions. In addition, even at large scale and average degrees ranging from 5 to 20, the resulting DAGs have a “rich structure” and do not collapse to bipartite graphs.

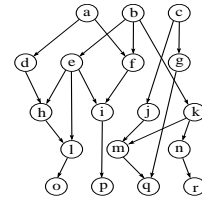


Fig. 4: A sample DAG

II. EXISTING GRAPH GENERATORS

The simplest graph generator is due to Erdos-Renyi [10] which generates a random graph. Let n, p be the number of

¹Our interpretation of rich structure is any attribute that is of higher complexity than number of nodes and edges. Such attributes include degree distribution, path-length distribution IV-0a

nodes and the edge probability, respectively. The generator by Erdos-Renyi creates a graph with n nodes and (expected) np number of edges. An edge is created by picking two nodes at random and joining them. They show that even such a simple generator exhibits an interesting phenomena of phase change. Namely, there exists a narrow range for p values for which the number of connected components drop significantly for very small increment in p . Planar triangulated graphs [17] are another class of random graph generators in which the points are embedded on a Euclidean plane. The Delaunay triangulation of the points yields a random graph. The R-MAT or Kronecker product based graph generators, mentioned in the introduction, is by far the most widely used graph generator. It has been extensively used to generate scale free graphs. All of these generators are meant for creating random or scale free, directed or undirected graphs.

In [20], the authors presented a linear programming based approach for generating acyclic graphs. In their approach, each node is a variable in the linear programming formulation. While the graphs generated using this approach are rich, a significant limitation of this algorithm is that it does not scale to multi-millions node acyclic graphs due to the extensively large size of the linear programming systems involved. The other difficulty with this approach is that it consists of many constraints. Setting up the constraints such that a solution to the linear program exists is non-trivial. Another approach for generating acyclic graphs was presented in [18], where the authors strove to generate acyclic graphs with given number of nodes uniformly at random. Their goal was to develop test suites for graph drawing packages and focused mostly on graphs of sizes in the range of 100–1000 nodes. The limited dataset size feasible through this procedure hints that purely random generation of acyclic graph is computationally very expensive.

The most popular scheme for generating acyclic graphs is to start with a simple acyclic structure (e.g. chains or binary tree) and add edges between nodes (u, v) if the level of u is less than the level of v . The nodes u, v in the node pair are selected randomly. As mentioned earlier in section I, graphs generated using such techniques, though scalable, have an inherent tendency to collapse. In the rest of this section, we pause briefly introduce notations (to be used throughout the paper) and then proceed to discuss the limitations of random acyclic generator.

A. Notations, Terminologies, and, Definitions

Given a directed graph $G = (V(G), E(G))$, let $n = \#V(G)$ be the number of nodes and $E(G)$ denote the number of edges in G .

A path is a sequence of edges,

$$\vec{P} = \langle (v_1 = u, v_2), (v_2, v_3) \dots, (v_{t-2}, v_{t-1}), (v_{t-1}, v_t = v) \rangle.$$

Alternatively, the same path can be represented as a sequence of vertices $\vec{P} = \langle v_1, v_2, \dots, v_t \rangle$. We say that a path is simple if a node appears at most once in the sequence; else the path has cycles. Graph G is a **directed acyclic graph** (DAG) if all paths are simple. Hence, nodes in acyclic graphs naturally

induce a topological ordering. Let $T(u)$ denote the rank of node u induced by the topological ordering.

An edge $e \in E(G)$, in acyclic graphs, is said to be out-incident on node u and in-incident on node v if $e = (u, v)$. Further, u is termed as an **in-incident** neighbor (or parent) of v and v is termed as an **out-incident** neighbor (or children) of u . The total number of in-incident neighbors of a node u is termed as its in-degree, denoted by $I(u)$. Similarly, the total number of out-incident neighbors of u is termed as its **out-degree**, denoted by $O(u)$. Nodes with in-degree zero are termed as root nodes while nodes with out-degree zero are leaf nodes. We call the rest of the nodes as DAG nodes. The rest of the notations will be defined as and when required during algorithm description. Figure 4 shows an example DAG with $parents(e) = \{b\}$ and $children(e) = \{h, l, i\}$.

B. Collapsing Nature of Acyclic Graphs

With very high probability random acyclic generators yield bipartite graphs irrespective of the initial structure and the type of random number generator. We demonstrate this phenomena when the initial structure is a binary trees. Let B_T be a binary tree with n nodes and $n - 1$ edges. Let td_v denote the tree depth of node v in this tree. We follow the convention that root is at level 0. The level is in ascending order along the child/descendant axis. To this tree we add $cn(1 + \epsilon)$ edges for some fixed c and $0 < \epsilon < 1$, resulting in average degree of the generated graph to be $c(1 + \epsilon)$. Each edge $e = (u, v)$ is directed from the source u to v if $td_u < td_v$. Let dd_v denote the depth of the node v in the graph *after* addition of random edges, dd_v being defined as

$$dd_v = \min\{dd_u + 1 | u \text{ is-a parent of } v\}. \quad (1)$$

Lemma 2.1: Let (u, v) be randomly selected pair under the constraint that $td_u < td_v$. Let E_v denote the event that $td_u < td_v - 1$. Then $Pr(E_v) = \frac{1}{2}$

Proof: Let k be the number of nodes in the tree at depth = $td_v - 1$. Then, the total number of nodes with depth $< td_v - 1$ is $k - 1$ (by property of binary trees). This implies that $P(td_u < td_v - 1) = P(td_u = td_v - 1) = \frac{k}{2k-1} = 1/2$, for all practical purposes. Hence, $P(E_v) = \frac{1}{2}$. ■

Suppose the generation is performed in c iterations and in each iteration $n(1 + \epsilon)$ edges are added. The source and destination nodes are selected randomly. We say that a node is *selected* in a iteration if it is a target of any of the $n(1 + \epsilon)$ randomly selected edges.

Claim 2.1: In a iteration, with high probability (w.h.p) almost every node selected as a target of a randomly selected edge.

Proof: Let \bar{X}_γ denote that γ nodes are not selected as target during an iteration. We are required to show that $Pr[\bar{X}_\gamma]$ approaches 0 from some small value of γ .

Consider a node u . Let X_u^i denote the event that the node is selected as target node for the i^{th} randomly generated edge. Let $X_u = \sum_{i \in [1:(1+\epsilon)n]} X_u^i$ denote the event the u is selected *at least* once as destination and \bar{X}_u denote the event that u is never selected target i.e $Pr[\bar{X}_u] = 1 - Pr[X_u]$.

Since there are n possible choices for choosing a target node it follows that $Pr[X_u^i] = 1/n$ and $Pr[\bar{X}_u^i] = 1 - 1/n$. Hence,

$$Pr[\bar{X}_u] = \prod_{i \in [1:n(1+\epsilon)]} Pr[\bar{X}_u^i] = (1 - 1/n)^{n(1+\epsilon)} \quad (2)$$

The probability that a given set of γ nodes, $u_1, u_2, \dots, u_\gamma$ are never chosen as target is

$$\prod_{j \in [1:\gamma]} Pr[\bar{X}_{u_j}] = (1 - 1/n)^{n\gamma(1+\epsilon)} \quad (3)$$

Since there are $\binom{n}{\gamma}$ ways of selecting γ points, the probability that γ nodes remain unselected comes out to be:

$$\binom{n}{\gamma} (1 - 1/n)^{n\gamma(1+\epsilon)} \quad (4)$$

Simplifying the equation using elementary estimates we have

$$\begin{aligned} Pr[\bar{X}_\gamma] &= \binom{n}{\gamma} (1 - 1/n)^{n\gamma(1+\epsilon)} \\ &< \left(\frac{en}{\gamma}\right)^\gamma e^{-\frac{1}{n}n\gamma(1+\epsilon)} \\ &< \left(\frac{en}{\gamma}\right)^\gamma \frac{1}{e^{\gamma(1+\epsilon)}} \\ &< \frac{n^\gamma}{\gamma^\gamma} \frac{1}{e^{\gamma\epsilon}} \end{aligned} \quad (5)$$

Assuming $\gamma = \epsilon = \log n$ we get

$$\begin{aligned} \binom{n}{\gamma} (1 - 1/n)^{n\gamma(1+\epsilon)} &< \frac{n^{\log n}}{\log n^{\log n}} \frac{1}{e^{\log n \log n}} \\ &< \frac{n^{\log n}}{\log n^{\log n}} \frac{1}{n^{\log n}} \\ &< \frac{n^{\log n}}{\log n^{\log n}} \frac{1}{n^{\log n}} \\ &< \frac{1}{\log n^{\log n}} \end{aligned} \quad (6)$$

The above derivation states that if in each round $n \log n$ edges are added then the probability that more than $\log n$ nodes are not assigned new edges is less than $O(\frac{1}{\log n^{\log n}})$. Since this expression quickly approaches zero we conclude that w. h.p. almost all nodes will be the target for at least one of the newly inserted edge.

An alternative bound for this can be obtained by applying Chernoff bound [5] on $\bar{X} = \sum_{u \in V} \bar{X}_u$. Since $Pr[\bar{X}_u] = (1 - 1/n)^{n(1+\epsilon)}$ it follows that the expected value μ of \bar{X} is $n * Pr[\bar{X}_u] = n(1 - 1/n)^{n(1+\epsilon)}$. By, Chernoff inequality

$$Pr[\bar{X} > \mu(1 + \delta)] < \left(\frac{e^\delta}{(1 + \delta)^{1+\delta}}\right)^\mu$$

Let $(1 + \delta) = \beta = \log \log n$ and let $\epsilon = \log n - 1$. Again, using basic estimates we obtain

$$\begin{aligned} \mu(1 + \delta) &= n(1 - 1/n)^{n(1+\epsilon)}(1 + \delta) \\ &< ne^{-(1+\epsilon)\beta} \\ &< ne^{-\log n \beta} \\ &< \log \log n \end{aligned} \quad (7)$$

Focusing on the RHS side of the Chernoff expression

$$\begin{aligned} \frac{e^{(\beta-1)\mu}}{\beta^\beta} &< \frac{e^{\beta-1}}{\beta^\beta} \\ &< \frac{e^{\log \log n}}{e^{\log \log n \log \log n}} \\ &< \frac{\log n}{e^{\log \log n \log \log n}} \end{aligned} \quad (8)$$

In other words, if $n \log n$ new edges are added to the graph then the probability that the number of unselected nodes is more than $\log \log n$ is less than $\frac{\log n}{e^{\log \log n \log \log n}}$.

Claim 2.2: With high probability $dd_v < \max(2, \frac{td_v}{2^c})$.

Proof: (Sketch) Consider B_T with $n = 2$. This tree has only two nodes, one being the root, the other being the leaf. Hence, adding a new edge does not induce any reduction in depth. Therefore,

$$dd_v = td_v = 1 < 2. \quad (9)$$

Now we consider B_T with $n > 2$. Through claim 2.1 we know that if in each iteration $n(1 + \epsilon)$ edges are added, where $\epsilon > \log n - 1$ then almost every node with high probability (w.h.p) is a target node in at least one of the inserted edges. Let c such iterations be performed. Let v be such a “selected” node. Let u be the corresponding new parent. Next, from equation (1), $dd_v \leq dd_u + 1$. By construction,

$$dd_u \leq td_u. \quad (10)$$

Using lemma (2.1) and equation (10), we get,

$$dd_v \leq td_u + 1 < (td_v - 1) + 1 = td_v \text{ with probability } \frac{1}{2}. \quad (11)$$

That is,

$$dd_v < td_v \text{ with probability } \frac{1}{2}. \quad (12)$$

Equation (12) says that every time an edge is added to a node, the node suffers at least a unit decrease in depth with probability $1/2$. Since there are $\log n$ edges per node (on average) we can derive a better bound on the probability that the node will suffer depth reduction. This is because for the node to not suffer depth reduction all the edges should have source node from depth $td_v - 1$. The probability of this is $1/2^{\log n} = 1/n$. Alternatively, we can say that with probability $1 - 1/n$ the node suffers a depth reduction.

Furthermore since (almost) every edge suffers a depth reduction it follows that the DAG depth of node v would be at least half of td_v i.e $dd_v < td_v/2$ at the end of the iteration with probability at least $1 - 1/n$.

Applying c iterations of edge insertions, will give

$$dd_v < td_v/2^c \text{ with probability } (1 - 1/n)^c. \quad (13)$$

Equations (9) and (13) together yield the claim. ■

Since the depth of a binary tree is $\log n$, if we perform merely $\log(\log n)$ iterations, we get $dd_v < \frac{td_v}{2^{\log \log n}} < \frac{\max td_v}{2^{\log \log n}} = \frac{\log n}{\log n} = 1$. That is, the entire tree (but for $\log \log n^{\log \log n}$ nodes w.h.p) will collapse to a single depth bipartite graph with probability $O(1 - 1/n)^{\log \log n} < O(e^{-\log \log n/n}) < 1/\log n^{1/n}$

In section IV-A we verify this phenomena for graphs generated using [13], which is a random acyclic graph generator similar to the one described above, except that they start with random sequences of nodes (instead of a tree) and add random edges of form (u, v) if the rank of u is less than the rank of v . We show using various DAG characteristics that the resulting graph collapses to depth of less than 3 and has a very “regular structure” when compared to DAG generated using our approach.

The collapsing phenomena, demonstrated here for binary trees, occurs irrespective of the initial shape of the tree.

Benchmarks based on such generators would simply test set-membership and set intersections and would not stress any graph centric capabilities of the applications, kernels and algorithms. Many real world acyclic graphs have average degree of more than 1. For e.g., citation datasets such as arxiv (from arxiv.org), and ,pubmed (from <http://www.ncbi.nlm.nih.gov/pmc/>), semantic knowledge database from www.mpi-inf.mpg.de/yago-naga/yago/, and, gene ontology terms along with their annotation files www.uniprot.org have average degrees of 11.12, 4.45, 6.38, and 4.99, respectively [21]. Studying these databases, therefore, calls for a different graph generating algorithm.

An alternative approach for producing acyclic graphs would be to modify or post process existing random generator and R-MAT generator to produce acyclic graphs. It is feasible to do so by finding strongly connected components (SCC) and reducing each connected component into a single node.

Claim 2.3: Acyclic graphs produced using SCCs have very few nodes irrespective of the size of the original digraph.

Proof: Let C_1, C_2, \dots, C_c be the maximally strongly connected components (SCCs) of a digraph G . A component of graph is strongly connected if there is a path from every node to every other node.² We create a DAG D with the set of new nodes $V(D) = \{v_1, v_2, \dots, v_c\}$, where each node v_i is the node produced by reducing the SCC C_i . Edges in D result from edges between distinct SCCs. Specifically, let $E(D)$ be the set of all edges in D . Then, $\exists(v_i, v_j) \in E(D)$ iff there is an edge $(c_i, c_j) \in E(G)$, where $c_i \in C_i$ and $c_j \in C_j, i \neq j$. We call D the reduced-DAG of digraph G . Typically, graphs produced by random and R-MAT generators have one giant and possibly few other SCCs [7]. (This phenomena is referred as “six degrees of separation” in social sciences and popular media.) Consequently, $V(D)$ is a very small set, and so is $E(D)$. Hence, any acyclic graph created starting with the graphs produced from these generators would have very few nodes irrespective of the size of the original digraph. ■

The next section describes the proposed scalable acyclic graph generator. The algorithm is designed with the idea of allowing more control on the characteristics of the output graph and allowing scalability at the same time.

III. GENERATING LARGE ACYCLIC GRAPHS

Our algorithm takes the number of nodes n , the desired depth d and four distribution functions as input — the fan-out, fan-in, slot and slot depth distributions. Notice that these distribution functions control the shape and characteristics of a graph. We denote these distributions by X_o, X_i, X_s , and, X_{sd} , respectively. Let x_o, x_i, x_s , and x_{sd} be the outcomes from their respective distributions on an invocation, i.e., $x = X()$.

The algorithm begins by creating n nodes and max_s slots where $max_s \gg d$. The n nodes are distributed over the max_s slots based on the slot distribution criteria X_s . We use S_i to denote the list of nodes shelved at slot i . Nodes at slot 0 form the roots of the DAG and nodes at slot s form the leaves. A node in any other slot can be a root, a leaf, or, a DAG-node.

Once the nodes have been slotted, we pick nodes starting from the topmost (low slot number) slot and build its out-edges. Suppose we are at slot i and building out-edges for node $u_i \in S_i$. We select $j > i$ based on slot depth distribution criteria X_{sd} , i.e. $j = i + X_{sd}()$. Then, from slot j , we randomly choose a node u_j and introduce an edge (u_i, u_j) . With the introduction of this edge, u_i has one out-degree satisfied and u_j has one in-degree satisfied. We repeat this process of selecting $u_j, j > i$, $O(u_i)$ number of times to satisfy all the out-degree requirements of u_i . Our implementation guarantees that each iteration picks a unique out-incidence node for u_i through the use of function *rpick_uniq*. This function guarantees that u_j is not present previously among the adjacent nodes of u_i . The in-degree of u_j must also be satisfied. To this end, we place $I(u_j)$ copies of u_j in slot j . Algorithms 1, 2, and 3 outline a broad description of the generator.

Let \mathcal{P} be permutation of numbers from $[0 : n]$ and $\mathcal{P}[i]$ denote the i^{th} value from this permutation. Algorithm 1 distributes the nodes into slots based on X_s and builds the permuted list for each slot. For each node u it invokes the slot distribution function $x_s = X_s()$ (line 2) and the in-degree distribution function $x_i = X_i()$. It places x_i copies of u in slot x_s (line 4–6). Finally, the algorithm permutes the nodes in each slot (line 8–10).

Let $adj(u)$ be the adjacencies (out-incident) nodes of u . Algorithm 2 builds the edges for a given node u . It first selects a slot $x_{sd} = X_{sd}()$ and then picks a node v from the list $S_{x_{sd}}$ to form directed edge $e = (\mathcal{P}[u], \mathcal{P}[v])$. The number of out-edges incident on u is sampled from degree distribution X_o .

Algorithm 3 builds the acyclic graph given the various distribution functions. It calls algorithm 1 to distribute nodes into slots. Once the nodes are distributed in their respective slots, it calls algorithm 2 for each node u , in ascending order of the slot, to build its out edges. This generates the desired acyclic graph.

Algorithm 1 *shelve_nodes*(X_s, X_i)

Require: max_s, x_s, X_i the maximum slot value, slot, and fan-in distributions respectively

```

1: for  $u \in [0 : n]$  do
2:    $x_s \leftarrow X_s()$ 
3:    $x_i \leftarrow X_i()$ 
4:   for  $j \in [0 : x_i]$  do
5:      $append(S_{x_s}, u)$ 
6:   end for
7: end for
8: for  $i \in [0 : D]$  do
9:    $random\_shuffle(S_i)$ 
10: end for
```

A. Choosing Distribution Functions

The choice of distribution functions affects the shape and various characteristics of the resulting DAG. Some combinations of distribution functions may not be feasible at all or may yield degenerate graph. Here we provide a set of distribution

²By definition, a single node is a SCC although not necessarily maximal.

Algorithm 2 *build_out_edges*(u, s, X_o, X_{sd})

Require: X_{sd}, \mathcal{P} the depth distribution and permutation vector respectively

```

1: for  $i \in [0 : X_o]$  do
2:    $x_{sd} \leftarrow X_{sd}()$ 
3:    $v \leftarrow \text{rpick\_uniq}(S_{x_{sd}+s})$ 
4:    $\text{append}(\text{adj}(\mathcal{P}[u]), \mathcal{P}[v])$ 
5: end for

```

Algorithm 3 *acyclic_generator*($max_s, X_s, X_i, X_{sd}, X_o$)

Require: X_s, X_i, X_{sd}, X_o as slot, input, depth, and output distributions respectively

```

1: Shelve_Nodes( $max_s, X_s, X_i$ )
2: for  $s \in [0 : max_s]$  do
3:   for  $u \in S_s$  do
4:     build_out_edges( $u, s, X_o, X_{sd}$ )
5:   end for
6: end for

```

functions that lead to non-degenerate graphs whose shapes can be controlled by tuning the parameters of the distribution function.

We consider Gaussian distribution with mean m and variance $\mu = m/5$ ($X = \mathcal{N}(m, \frac{m}{5})$), where m is the average degree of the desired DAG. (So, the total number of edges in the graph, $\#E(G) = mn$.) Figures in 5 display the Gaussian distribution (top row) for this set-up with $m = 5, 10, 15$, and 20. Gaussian distributions form good candidates for X_i and X_o . The slot distribution is set as the discrete geometric distribution with parameter p , i.e., $P(X = k) = (1 - p)^{k-1}p, 0 < p < 1$. The slot depth distribution may be chosen to be a scale free distribution and is parametrized by scale γ , i.e., $P(k) \approx ck^\gamma$. Figures 5 (bottom row) show the frequency distribution resulting from the geometric and scale free distributions for $p = 0.001$ and $\gamma = 0.04$, respectively.

The distribution functions presented here are in accordance with those used in other statistical and graph generators. The scale free distribution function used here is the same as the functions used in the R-MAT graph generator, just used differently. In R-MAT generator, this distribution is used as a joint probability distribution to pick both the source and the destination. This work uses this distribution for a fixed source to arrive at the slot of the target node. The target is then chosen randomly from the set of all nodes present in the target slot. It turns out that unlike X_i, X_o , and X_s , which we are free to choose, the slot depth distribution function X_{sd} should preferably be a scale free distribution so that the acyclic graph does not collapse. We conjecture that scale free distribution is central to the structure of the acyclic graph just as it was in the R-MAT graph generator for interaction graphs.

Correctness

We briefly discuss the correctness of the algorithm i.e. the proposed generators always will produce an acyclic graph. Our claim rests on the observation that if the graph is acyclic then

there exists a partial ordering \mathcal{PO} of the nodes such $\mathcal{PO}(u) < \mathcal{PO}(v)$ for all edges (u, v) of the DAG.

The slot depth distribution function X_{sd} naturally imposes such an ordering. For each node u , let $\mathcal{PO}(u)$ be the same as the depth of the slot it is assigned to. We know that the algorithm generates edge (u, v) by finding a node v under following constraints:

- u and v do not belong to the same slot depth
- Slot depth of u less than slot depth of v (line 2–4, Algorithm 2)

Both the constraints imply that for any generated edge (u, v) , $\mathcal{PO}(u) < \mathcal{PO}(v)$. Hence, the generated graph cannot contain any cycles.

Moreover, for small size graph we have verified through cycle detection algorithms that the generated graph does not contain any cycles. For large graphs we verify using the Erdős-Gallai theorem [6] which provides constraints of degree distribution of the nodes in a simple graph. It states that a sequence of non-negative integers $d_1 \geq d_2 \geq d_3 \geq \dots$ can be represented as finite simple graph on n vertices if and only if $d_1 + d_2 + d_3 + \dots + d_n$ is even and

$$\sum_{i=1}^k d_i \leq \sum_{i=k}^n \min(d_i, k) + k * (k - 1)$$

for all $k \in [1 : n]$. We have verified this constraint (for all values of k) for each of the large generated graphs.

IV. EXPERIMENTAL RESULTS

This section presents some of the experiments on the use of the proposed generator and the choice of distribution function. We generate a large acyclic graph and study its various characteristics. We validate our model across following three criteria:

- Ability to produce non-degenerate graphs as we scale the number of nodes and edges.
- The shape of the in- and out- degree frequency distribution of the resulting graph.
- The shape of the DAG depth distribution.
- The shape of the path length distribution discussed below.

a) Path Length Frequency Distribution: We characterize acyclic graphs based on the frequency distribution of the length of random walks starting from roots to the destination. The path length distribution characterizes the “richness” in the resulting structure. This attribute of acyclic graph is important because the depth parameter does not reveal the degree of irregularity. Two acyclic graphs with same depth value can have very different random path length frequency distributions. We can illustrate this by constructing a DAG $D = (V, E)$ where the nodes in V can be partitioned into sets V_1, V_2, \dots, V_t and the edges in E into E_1, E_2, \dots, E_{t-1} such that an edge $e = (u, v) \in E_i \implies u \in V_i$ and $v \in V_{i+1}$. This graph is a DAG with roots as V_1 and leaves as V_t and can be visualized as t bi-partite graphs stacked over each other. The depth of the graph is t . However, the path length frequency distribution contains only a single value t , i.e., every path in D has length t . Such a graph will not be very meaningful for benchmarking and characterization even though it may satisfy other acyclic graph properties.

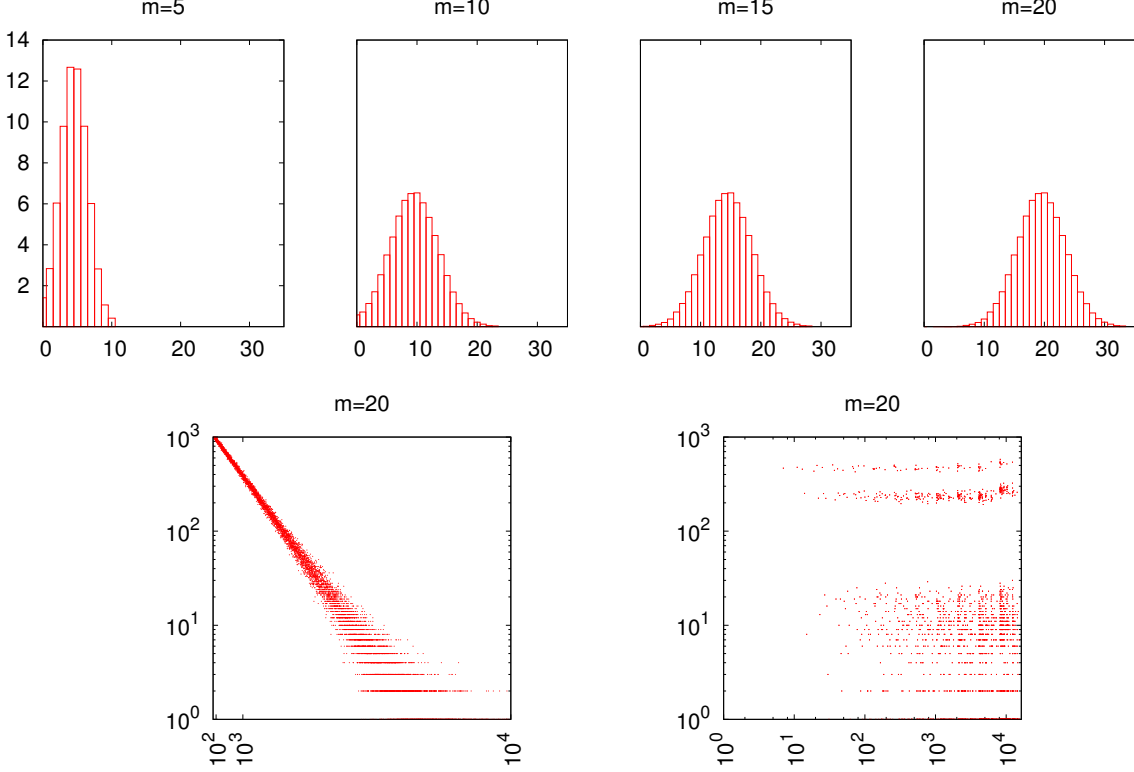


Fig. 5: Shapes of various distributions used in the generator. Top row illustrates Gaussian distributions (X_i, X_o) for various values of m . Bottom left is a plot for geometric frequency distribution $p = 0.001$ (X_s), while bottom right show scale free distribution $\gamma = 0.045$ (X_{sd}).

Graphs using our approach: While the generator has been able to generate graphs of size up to 2^{27} nodes, in this paper we display experiments with graphs of size $n = 2^{23}$ nodes. We experimented with various input distributions. For the experiments presented here, the Gaussian distributions suggested in section III are used for in-degree and out-degree distributions. For simplicity in presentation, we keep X_i same as X_o , i.e., $X_i = X_o = \mathcal{N}(m, \frac{m}{5})$. The values of p and γ for the discrete geometric and the scale free distributions are fixed $p = 0.001$ and $\gamma = 0.04$. Values of p less than this lead to degenerate acyclic graphs while values higher than this have little influence on the resulting depth distribution. We present results for different values of m . (Notice that the in- and out-degree distributions are independent of slot and slot depth distributions and allow freedom in choice even for fixed p, γ .)

Figures 6 (top row) show the resulting fan-out distribution for various values of m . Fan-out is directly controlled by variable X_o because the generator exactly creates $X_o()$ number of out-edges per chosen node u . Hence, as we vary m , the fan-out degree distribution of the output graph follows closely the input Gaussian distribution displayed in figures 5 (top row).

The fan-in distribution of the resulting DAG is shown in figures 6 (2nd row). Unlike out-degree, the in-degree of any node v is controlled by the input distribution X_i indirectly. Specifically, as explained in the algorithm description, the resulting fan-in degree distribution is derived by uniform sampling of X_i . Hence, the fan-in distribution shape of the

output graph is a slightly perturbed Gaussian.

Plots in third row of figure 6 demonstrate that the graph generated by the proposed algorithm does not collapse under increased degree. In fact, a comparison of the depth distributions for $m = 5$ and $m = 20$ shows that the maximum depth increases on increasing the number of edges in the graph.

We now turn our attention to the “structural richness” of the generated DAG. We measure this property by observing the distribution of random walks from source to leaves. A DAG with a rich structure would have a varied set of random path lengths. For the DAG generated by our generator, the path length distribution is displayed in figures 6 (last row). We see that the path length distribution follows a power law distribution. Moreover, adding more edges increases the structural richness of the graph, i.e., there is a wider variety of path lengths for greater number of edges. The steepness of the distribution (i.e., the slope of the exponent) decreases as we increase m .

b) Varying slot distribution function X_p : In the above experiments we choose the slot distribution function as discrete geometric distribution with parameter p , i.e., $P(X = k) = (1 - p)^{k-1}p$ where p was set to .001. The parameter p controls the depth attribute of the graph. We demonstrate this through illustration (using Graphviz) of graphs with varying value of p . The scale s of the graph is 9 while mean degree m is 5 with variance 1. Figures 7,8,9,10, and, 11 are drawings of the graph for $p = .03, .05, .08, .1$, and, .3 respectively. The width-to-height ratio in figure is same as determined by the

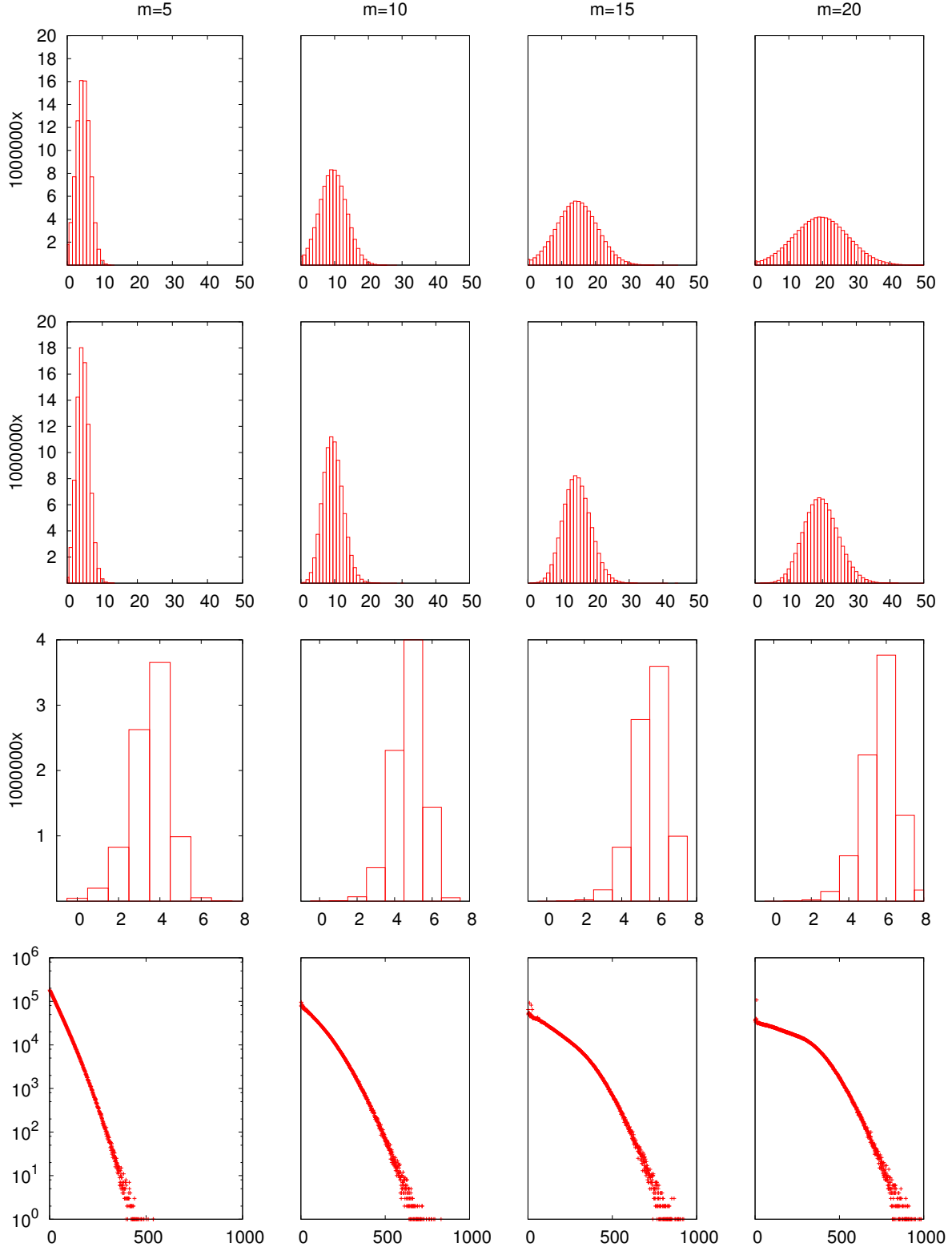
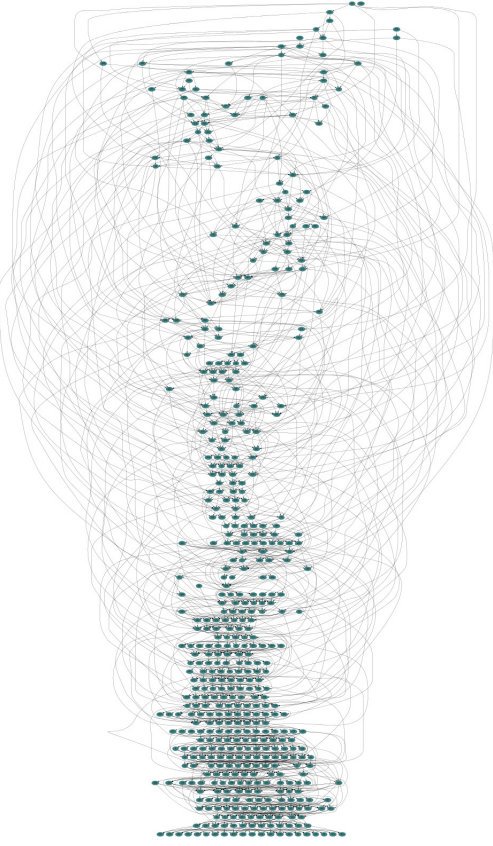
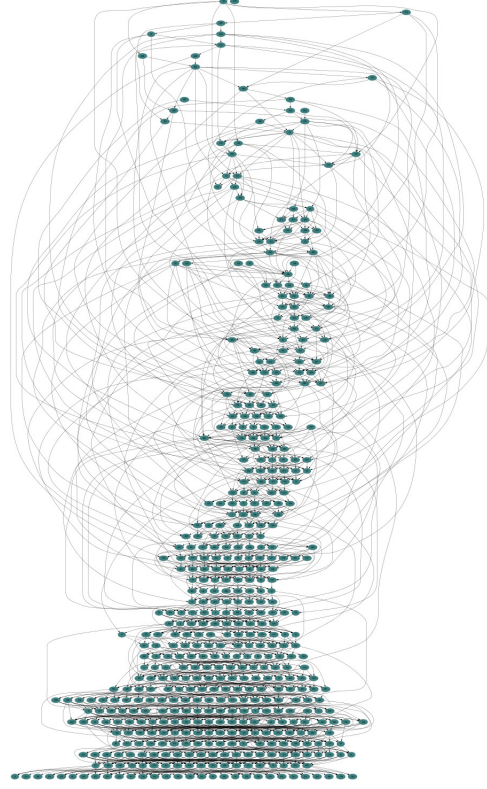
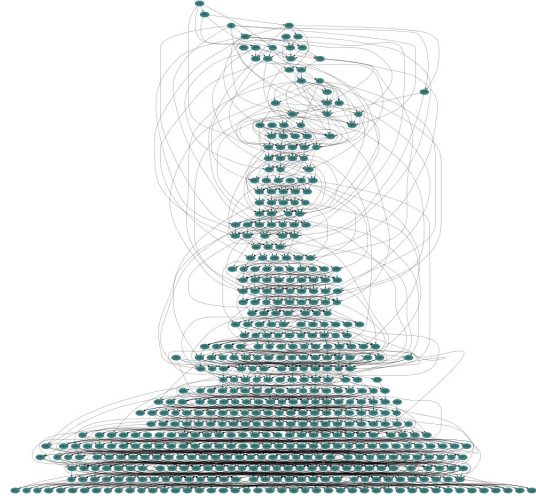
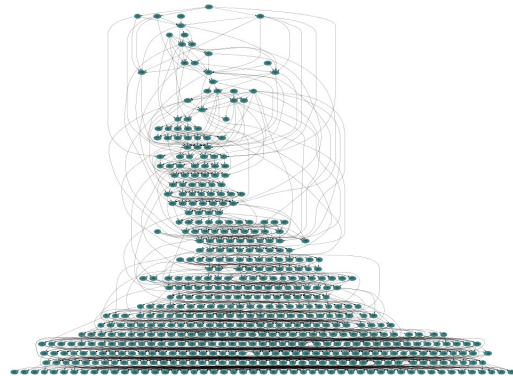


Fig. 6: Characteristics of the resulting DAG from our generator. Top row: fan-out degree closely follows the input. 2nd row: fan-in degree of the graph obtained from the generator is a perturbed version of the input. 3rd row: Depth distribution suggest collapse free structure. Bottom row: path length distributions imply high structural richness.

Fig. 7: $p = 0.03$ Fig. 8: $p = 0.05$

Graphviz program. We see that as we increase p the depth becomes smaller. Such control over the shape of dag is not feasible with the random generator.

c) Scalability of the generator: So far we have studied various characteristics of the graphs produced by the generator. We now present experiment to study the scalability of the generator itself. We study the time required to generate the graph as function of graph size and mean degree. Figures 12 and 13 show the results. In figure 12 we study the scalability with respect to mean degree for a fixed scale graph while in figure 13 we study the scalability with respect to the graph scale parameter but for a fixed mean degree. The y-axis represent time, in log scale, to generate the graph, while the x-axis denote mean degree and scale respectively for figures 12 and 13. As we can see the generation time growth is linear with respect to mean degree. However the slope of growth is higher at larger scale graphs. This is because the complexity of data structures use to maintain S_i , i.e., list of nodes at depth i and its associated operation, $rpick_uniq(S_i)$ for any depth i grows non-linearly with the scale of the graph. The generator also scales nicely with respect graph size, i.e., linear in log scale with respect to the scale, or, linear in absolute generation time with respect to the number of nodes in the graph assuming a fixed mean degree. The memory requirements of our algorithm is proportional to the requisite graph size.

Fig. 9: $p = 0.08$ Fig. 10: $p = 0.1$

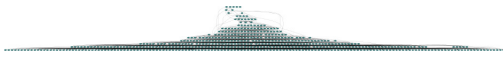
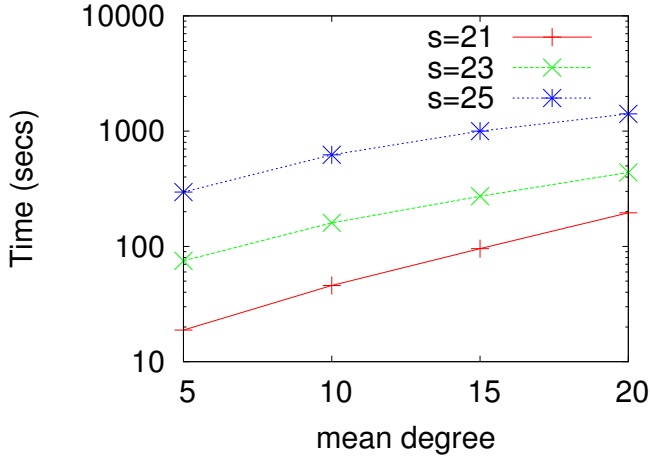
Fig. 11: $p = 0.3$ 

Fig. 12: The generator is linearly scalable with respect to mean degree. However, slope of the curve is higher for larger scale. This is because book keeping data structures for S_i scale non-linearly with the size of the graph.

A. Random Acyclic graphs

We illustrate the distributions obtained through a random acyclic graph generator to support some of the arguments presented in section II-B and to compare the characteristics of graph with those generated by our scheme. The number of nodes n and the number of edges m was used as the input to the generator. Figures 14 (top row) show the fan-in degree distributions (fan-out degree distributions are similar). Notice that the user can only specify the number of nodes and edges and does not have much control over the shape of the various degree distributions of the output DAG. In general, the fan-in degree distribution turns out to be an exponential distribution. Figures 14 (middle row) show the depth distribution. We see

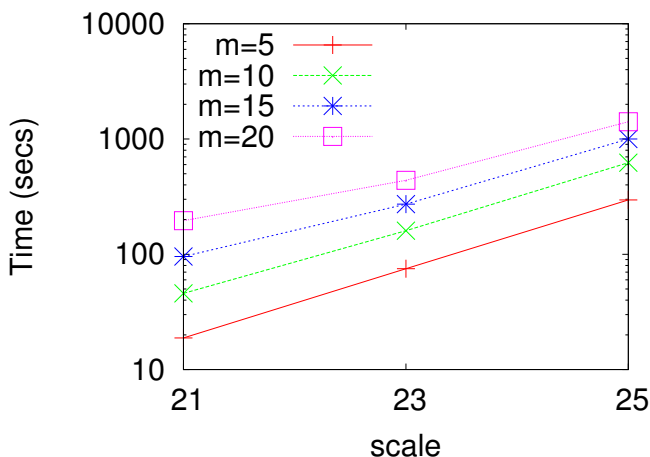


Fig. 13: The generator shows linear growth with respect to graph size.

that irrespective of the number of edges in the graph, the DAG depth collapses to a very small value (≤ 3). Figures 14 (bottom row) display the path length distribution. Again, here we see that there is not enough irregularity in the structure. Any random path from a root to a leaf is very highly likely to have path length of ≈ 12 . This is in quite contrast with the path length distribution generated through our scheme which shows a natural scale free distribution with path lengths ranging from 0 to 800.

V. CONCLUSION

Acyclic graphs used in literature to study the performance of many semantic and information centric operators are often generated using random selection of edges in an already existing tree. We analytically and experimentally show that such schemes produce degenerate graphs. Moreover, there is no control on the shape and structure of the resulting graph. The few schemes designed so far to generate non-degenerate graphs are not scalable. We present a scalable algorithm that relies on carefully selecting various distribution functions and their associated parameters so as to produce acyclic graphs. Scale free distribution turns out to be crucial to maintaining the non-degeneracy of directed acyclic graphs. Via the input parameters, the user has much control over the graph to be generated. The richness in the overall DAG characteristics (output dags) is seen in properties such as it's binomial depth distribution, Gaussian degree distribution, and scale free path length distribution. Furthermore, richness in the structure of the generated DAG is verified using the variety in path lengths (path length distribution). Our experiments yield positive and encouraging verification of all these properties. An interesting future problem is to find the right parameters for the generator that will model given real world DAGs. We are also looking into the problem of generating labeled DAGs to model ontology data coming out of various life sciences discipline.

REFERENCES

- [1] A.-L. Barabasi and R. Albert. Emergence of scaling in random networks. *Science*, 286:509, 1999.
- [2] R. Bramandia, B. Choi, and W. K. Ng. Incremental maintenance of 2-hop labeling of large graphs. *IEEE Trans. on Knowl. and Data Eng.*, 22(5):682–698, May 2010.
- [3] D. Chakrabarti, Y. Zhan, and C. Faloutsos. R-mat: A recursive model for graph mining. In M. W. Berry, U. Dayal, C. Kamath, and D. B. Skillicorn, editors, *SDM*. SIAM, 2004.
- [4] J. Cheng, J. X. Yu, X. Lin, H. Wang, and P. S. Yu. Fast computing reachability labelings for large graphs with high compression rate. In *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*, EDBT '08, pages 193–204, New York, NY, USA, 2008. ACM.
- [5] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, 23(4):493–507, 1952.
- [6] S. Choudum. A simple proof of the erdos-gallai theorem on graph sequences. *Bulletin of the Australian Mathematical Society*, 33(01):67–70, 1986.
- [7] R. Cohen and S. Havlin. Scale-Free Networks Are Ultrasmall. *Phys. Rev. Lett.*, 90:058701, 2003.
- [8] H. Dehainsala, G. Pierra, and L. Bellatreche. Ontodb: an ontology-based database for data intensive applications. In *Proceedings of the 12th international conference on Database systems for advanced applications*, DASFAA'07, pages 497–508, Berlin, Heidelberg, 2007. Springer-Verlag.

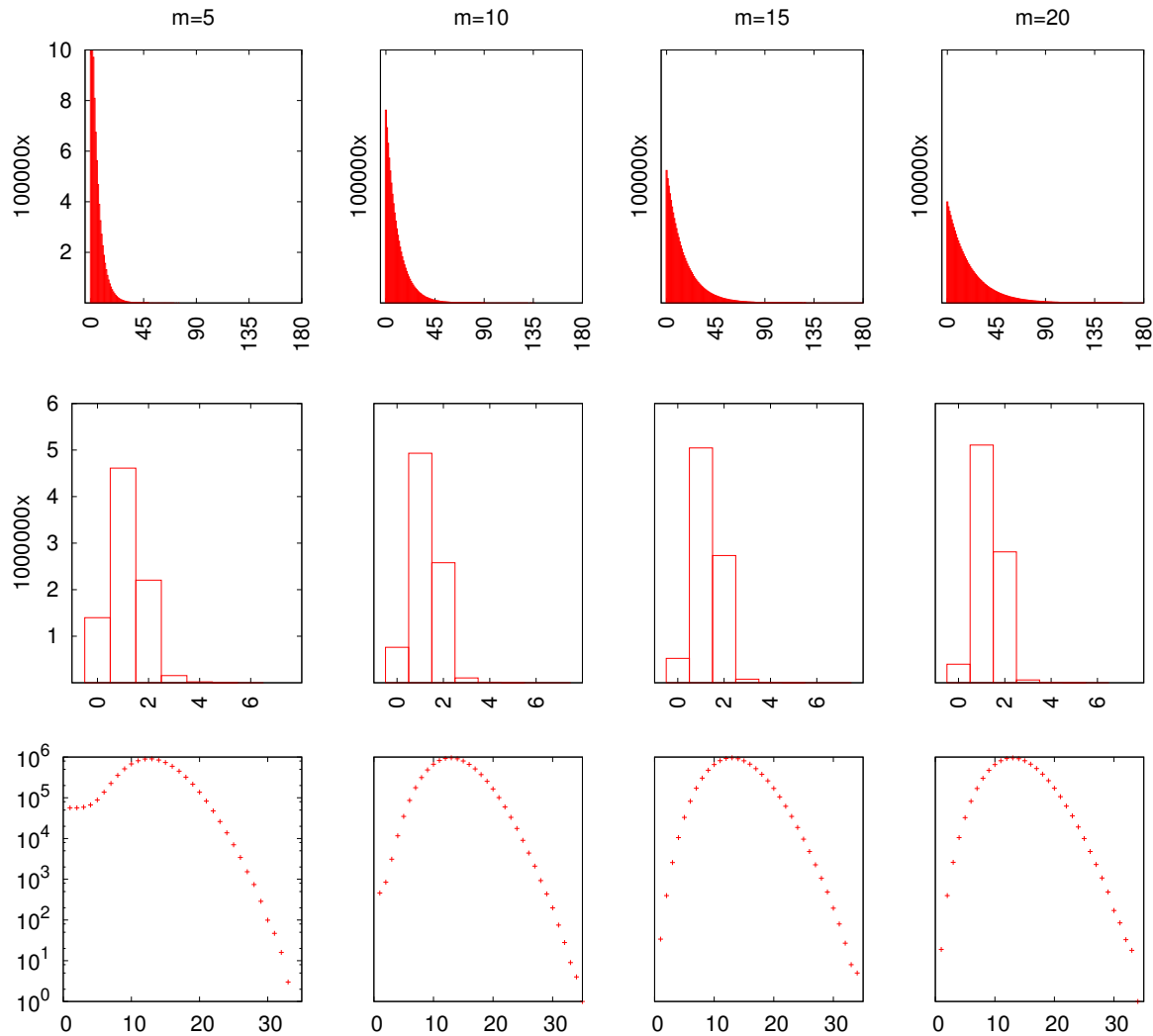


Fig. 14: Random DAG generator: Fan-in degree of the graph is unchanged for different inputs. Depth degree distribution of the graph remains same for all inputs and is mostly focused around three values. High concentration of path length distribution in small range of 15 – 25 implies predictable structure.

- [9] J. Ellson, E. R. Gansner, E. Koutsofios, S. C. North, and G. Woodhull. Graphviz - open source graph drawing tools. In P. Mutzel, M. Jnger, and S. Leipert, editors, *Graph Drawing, 9th International Symposium, GD 2001 Vienna, Austria, September 23-26, 2001, Revised Papers*, volume 2265 of *Lecture Notes in Computer Science*, pages 483–484. Springer, 2001.
- [10] P. Erdos and A. Renyi. On the evolution of random graphs. *Publ. Math. Inst. Hungar. Acad. Sci.*, 5:17–61, 1960.
- [11] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication, SIGCOMM '99*, pages 251–262, New York, NY, USA, 1999. ACM.
- [12] R. Jin, Y. Xiang, N. Ruan, and H. Wang. Efficiently answering reachability queries on very large directed graphs. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data, SIGMOD '08*, pages 595–608, New York, NY, USA, 2008. ACM.
- [13] R. Johnsonbaugh and M. Kalin. A graph generation software package. In *Proceedings of the twenty-second SIGCSE technical symposium on Computer science education, SIGCSE '91*, pages 151–154, New York, NY, USA, 1991. ACM.
- [14] J. Köhler, S. Philippi, and M. Lange. Semeda: ontology based semantic integration of biological databases. *Bioinformatics*, 19(18):2420–2427, 2003.
- [15] R. Kumar, P. Raghavan, S. Rajagopalan, D. Sivakumar, A. Tomkins, and E. Upfal. Stochastic models for the web graph. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 57–, Washington, DC, USA, 2000. IEEE Computer Society.
- [16] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani. Kronecker graphs: An approach to modeling networks. *J. Mach. Learn. Res.*, 11:985–1042, March 2010.
- [17] S. Meinert and D. Wagner. An experimental study on generating planar graphs. In M. Atallah, X.-Y. Li, and B. Zhu, editors, *Frontiers in Algorithmics and Algorithmic Aspects in Information and Management*, volume 6681 of *Lecture Notes in Computer Science*, pages 375–387. Springer Berlin / Heidelberg, 2011.
- [18] G. Melancon, I. Dutour, and M. Bousquet-Melou. Random generation of dags for graph drawing. Technical report, Amsterdam, The Netherlands, The Netherlands, 2000.
- [19] H. Tangmunarunkit, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. Network topology generators: degree-based vs. structural. *SIGCOMM Comput. Commun. Rev.*, 32:147–159, August 2002.
- [20] Y. Theoharis, G. Georgakopoulos, and V. Christophides. On the synthetic generation of semantic web schemas. In V. Christophides, M. Collard, and C. Gutierrez, editors, *SWDB-ODBS*, volume 5005 of *Lecture Notes in Computer Science*, pages 98–116. Springer, 2007.
- [21] H. Yildirim, V. Chaoji, and M. J. Zaki. Grail: scalable reachability index for large graphs. *Proc. VLDB Endow.*, 3:276–284, September 2010.